DEBUGGER IMPACT REDUCTION THROUGH MOTION
OF INDUCTION VARIABLE BASED BREAKPOINTS

Cross Reference to Related Applications

5   The following is a related co-pending United States patent application:

DEBUGGER IMPACT REDUCTION THROUGH BREAKPOINT MOTION, Serial

No. xx/xxx,xxx, filed July 3, 2001, by Cary Lee Bates and William Jon Schmidt.

Background of the Invention

10   1.   Field of the Invention

The invention relates to debugger impact reduction through motion of

induction variable based breakpoints.

2.   Background Information

15   A debugger is a software tool used by programmers in software development

to identify and correct program errors within a program. To support the debugger,

information describing symbols and types in the program as well as information to

map between source lines and the binary code may be provided by a compiler and/or

an interpreter ("compiler"). This extra information, generally referred to as debugging

20   information, enables the programmer to examine the types, variables and data

structures by name and to follow the execution of the program through the source

code.

With a debugger, programmers may "step" through program code statements

one at a time, while the corresponding machine instructions are being executed. As

25   programmers step through code, they may, inter alia, concurrently monitor changes in

certain variables, which assists in identifying and correcting program errors.

With a debugger, a programmer may also set a breakpoint at a specific program position. When the program is executed, the debugger will stop execution of the program at any breakpoint encountered, and may display various programming elements, such as software variables for example, to assist in the debugging process.

5   Debuggers may also allow programmers to define and set conditional breakpoints. A conditional breakpoint is a breakpoint that has an associated Boolean expression. Upon encountering a conditional breakpoint, a debugger will only stop execution of the program if the associated Boolean expression is evaluated by the debugger to be TRUE (i.e., satisfied). For example, a conditional breakpoint may be

10   BREAK 6 WHEN K < 999. Accordingly, a debugger would only stop execution of the program at the location of such a conditional breakpoint (here, line 6) if the variable K is less than the value 999 when the conditional breakpoint is encountered.

In today's software development environment, many new software applications are threaded. As such, they are designed to efficiently execute in

15   multitasking or multiprocessing environments, which allow multiple threads, or streams of execution, to execute concurrently. With the increased frequency of threaded applications, programming timing errors are becoming more common than ever.

A significant problem with debugging program timing errors is that the

20   introduction of breakpoints may dramatically affect program timing, which may make difficult or impossible the reproduction of program errors in the debugging environment. While conditional breakpoints may affect overall program timing less than normal breakpoints and program code stepping, they still require a short amount of time for their associated Boolean expressions to be evaluated. Although this short

25   amount of time may be tolerable in some cases, where the conditional breakpoint is

encountered a large number of times, program timing may be so significantly affected

that reproducing program errors in a debugging environment may be difficult or

impossible.

A frequently occurring situation in which a conditional breakpoint may be

5   encountered a large number of times is where the breakpoint is set within a program

loop. A program loop is a repetition within a program, and may iterate a large

number of times. Therefore, where a conditional breakpoint is set within a program

loop, program timing is more likely to be substantially affected, and accordingly, the

difficulty or impossibility of reproducing program errors in the debugging

10   environment becomes more likely. Further, a conditional breakpoint set within a

program loop may be based on the induction variable that also controls the program

loop (i.e., an induction variable based breakpoint).

Therefore, it would desirable to provide debugger impact reduction through

motion of induction variable based breakpoints.

15

Summary of the Invention

It is, therefore, a principle object of this invention to provide debugger impact

reduction through motion of induction variable based breakpoints ("IV-breakpoints").

It is another object of the invention to provide debugger impact reduction

20   through motion of IV-breakpoints that assists in minimizing the above mentioned

problems.

These and other objects of the present invention are accomplished by

debugger impact reduction through motion of IV-breakpoints as disclosed herein.

In an exemplary aspect of the invention, a method of reducing debugger

25   impact through motion of an IV-breakpoint set within a program loop, where the IV-

3

breakpoint and the loop are controlled by an induction variable having an induction

rate, may comprise: extracting, from program code within the program loop, the

induction rate; extracting, from the IV-breakpoint, a final value of the induction

variable for which the IV-breakpoint would be satisfied; and if the IV-breakpoint is

5   satisfied and the induction variable has a present value that would be beyond the final

value upon a next iteration of the loop based on the induction rate, removing the IV-

breakpoint.

In a further exemplary aspect of the invention, a method of reducing debugger

impact through motion of an IV-breakpoint may further comprise: setting, at one or

10   more loop exit program positions, a reset breakpoint; and if one of the reset

breakpoints is satisfied, removing the reset breakpoints and/or reestablishing the IV-

breakpoint.

Each of the above aspects may be embodied in a method, a debugger, and an

article of manufacture comprising a computer readable program storage medium

15   tangibly embodying one or more programs of instructions executable by a computer.

Brief Description of the Drawings

Figure 1 is a flow diagram illustrating debugger impact reduction through

motion of an IV-breakpoint, set within a program loop, with optional reset breakpoint

20   settings, according to an embodiment of the invention.

Figure 2 is a flow diagram illustrating debugger impact reduction according to

another embodiment of the invention.

Figure 3 illustrates an exemplary set of program code used to describe the

embodiments of Figures 1-2 in operation.

4

Figure 4 illustrates another exemplary set of program code used to describe the embodiments of Figures 1-2 in operation.

Detailed Description of the Preferred Embodiments

5   The invention will now be described in more detail by way of example with reference to the illustrative embodiments shown in the accompanying figures. It should be kept in mind that the following described embodiments are only presented by way of example and should not be construed as limiting the inventive concept to any particular configuration or order.

10   As shown in Figures 1 and 2, in exemplary aspects of the invention, a debugger (not shown) may effectuate IV-breakpoint motion through an induction rate extraction (block 2); a final value extraction (block 4); an optional reset breakpoint setting (block 6); an IV-breakpoint removal (block 8); an optional reset breakpoint removal (block 10); and an optional IV-breakpoint reestablishment (block 12).

15   Reference is now made to Figures 3 and 4 to illustrate exemplary breakpoint motions by a debugger (not shown) according to aspects of the invention as illustrated in Figures 1 and 2. Figures 3 and 4 illustrate exemplary sets of program code containing exemplary IV-breakpoints within exemplary program loops.

As shown in Figures 3 and 4, an IV-breakpoint 20 may be set within a

20   program loop, where the IV-breakpoint 20 and loop are controlled by an induction variable 22. In Figure 3, IV-breakpoint 20 is set within a for-loop, at line 6, and induction variable 22 is the variable "J"; while in Figure 4, IV-breakpoint 20 is set within a while-loop, at line 6, and induction variable 22 is also the variable "J".

In an induction rate extraction (block 2), a debugger may extract, from program code within the program loop, an induction rate 24 of induction variable 22. An induction rate extraction (block 2) may take place before IV-breakpoint 20 is satisfied, as illustrated in Figure 1, or after IV-breakpoint 20 is satisfied, as illustrated

5   in Figure 2.

Thus, as shown in Figure 3, a debugger may extract induction rate 24 as "J=J+1" from the increment expression "J++" on line 4 either before or after IV-breakpoint 20 is satisfied. Notably, induction rate 24 need not be a positive rate. For example, induction rate 24 may be "J=J-2" for a particular program loop.

10   As illustrated in Figure 4, induction rate 24 may be variable, as long as induction rate 24 is determinable by the debugger at some point up to when IV-breakpoint 20 is satisfied. Thus, as shown in Figure 4, a debugger may extract induction rate 24 as "J=J+1" from the increment statement "J++" on line 8 after evaluating the variable "FLAG" to be true; or as "J=J+2" from the increment

15   statement "J+=2" on line 10 after evaluating the variable "FLAG" to be false, both either before or after IV-breakpoint is satisfied.

In a final value extraction (block 4), a debugger extracts from IV-breakpoint 20 a final value 26 of induction variable 22 for which IV-breakpoint would be satisfied, either before IV-breakpoint is satisfied, as shown in Figure 1, or after IV-

20   breakpoint is satisfied, as shown in Figure 2.

As illustrated in Figure 3, a debugger extracts final value 26, from the condition "J<500" of IV-breakpoint 20, as the value "499" since induction rate 24 has a positive rate and the value "499" is the last value of induction variable 22 for which IV-breakpoint 20 may be satisfied. As illustrated in Figure 4, a debugger extracts

25   final value 26, from the condition "J<500 OR J==600" of IV-breakpoint 20, as the

value "600" since induction rate 24 has a positive rate and the value "600" is the last possible value of induction variable 22 for which IV-breakpoint 20 may be satisfied.

It should be noted that final value 26 may be either a technically exact last value for which a respective IV-breakpoint 20 would be satisfied, or simply the last possible value for which a respective IV-breakpoint may be satisfied. For example, in Figure 3, were induction rate 24 "J=J+2", the technically exact last possible value of induction variable 22 would be the value "498", whereas the last possible value would be "499". However, extracting either value as final value 26 would achieve the same result, which will be evident from the discussion below.

It should be noted that a program loop may be encountered only once during execution of the program. However, a program loop may be encountered multiple times during execution of a program. Therefore, the present invention includes setting one or more optional reset breakpoints to accommodate program loops that may be encountered multiple times.

In an optional reset breakpoint setting (block 6), the debugger may set a reset breakpoint 28 (discussed in further detail below) at one or more loop exit program positions. In Figure 3, the debugger sets a reset breakpoint 28 at line 8; while in Figure 4, the debugger sets a reset breakpoint at lines 12 and 14.

It should be noted that a loop exit program position may be a program position that would be reached after, or as, the loop finishes iterating, but nonetheless, before the respective loop is subsequently reentered. A loop exit position may be within the loop itself, as is illustrated in Figure 4, in which a reset breakpoint 28 is set within the loop at line 12; or outside the loop, as is illustrated in Figure 3, in which a reset breakpoint is set outside the loop at line 8.

In an IV-breakpoint removal (block 8), two initial conditions exist. First, the IV-breakpoint is satisfied; and second, the induction variable 22 has a present value that would be beyond the final value 26 upon the next iteration of the loop based on the induction rate.

5   Thus, as to the second condition, referring to Figure 3, after IV-breakpoint 20 is satisfied, the debugger may observe, in the 499th iteration of the loop, that induction variable 22 has a present value of "499", which due to induction rate 24 being "J=J+1", will have a value of "500" upon the next iteration of the loop, which is beyond final value 26 ("499"). Accordingly, the debugger will know that IV-

10   breakpoint 20 would not be satisfied for the remaining iterations of the loop, and thus, would remove IV-breakpoint 20.

Again as to the second condition, referring to Figure 4, after IV-breakpoint 20 is satisfied, the debugger may observe, in a particular iteration of the loop (which will depend on the value of the variable FLAG), that induction variable 22 has a present

15   value that due to induction rate 24 being "J=J+1" or "J=J+2" will have a certain value (i.e., "601" or "602") upon the next iteration of the loop that would be beyond final value 26 ("600"). Accordingly, the debugger would know that IV-breakpoint 20 will not be satisfied for the remaining iterations of the loop, and thus, will remove IV-breakpoint 20.

20   It should be noted that some debuggers may allow a programmer to change, during a debugging session, the value of a variable, which may be induction variable 22. Under such circumstances, if IV-breakpoint 20 had been removed, and may be thereafter satisfied due to the new value given to induction variable 22, optional IV-breakpoint reestablishment (block 12) (discussed further below) may be effectuated to

25   accommodate this situation.

After removing IV-breakpoint 20, the debugger creates a state that assists in avoiding unnecessary evaluations of IV-breakpoint 20 that may have occurred within the program loop. From this state, a programmer may debug the program as initially desired, and when ready, allow the program to further execute. When program

5   execution reaches a reset breakpoint 28, an optional reset breakpoint removal (block 10) may be effectuated, as the reset breakpoint 28 would no longer be needed. Also, when program execution reaches a reset breakpoint 28, an optional IV-breakpoint reestablishment (block 12) may be effectuated to allow IV-breakpoint 20 to be subsequently satisfied when the program loop is reentered.

10   It should be understood, however, that the invention is not necessarily limited to the specific process, order, arrangement and components shown and described above, and may be susceptible to numerous variations within the scope of the invention. For example, although the above-described exemplary aspects of the invention are believed to be particularly well suited for reducing debugger impact

15   with for-loops and while-loops, it is contemplated that the concepts of the present invention can be adapted for various types of programming loop and programming language syntaxes. For example, the concepts of the present application can be utilized with a do-while-loop, a goto-loop, etc., in Assembly, Basic, C, C++, etc.

Further, the debugger of the present invention may encompass other software

20   means insofar as the other software means perform an aspect of the present invention. For example, to the extent a compiler performs an induction rate extraction (block 2), the debugger of the present invention is intended to encompass that portion of the compiler.

It will be apparent to one skilled in the art that the manner of making and

25   using the claimed invention has been adequately disclosed in the above-written

9

description of the embodiments taken together with the drawings. Further, it will be apparent that the above methods may be readily embodied in a method, a software debugger, and a computer program medium, which may be any form of a computer program medium readable by a computer, such as, for example, a CD-ROM, RAM, a

5   hard drive, a cache, etc.

It will be understood that the above description of the embodiments of the present invention are susceptible to various modifications, changes, and adaptations, and the same are intended to be comprehended within the meaning and range of equivalents of the appended claims.